MODEL-BASED INTEGRATION TESTING TECHNIQUE USING FORMAL FINITE STATE BEHAVIORAL MODELS FOR COMPONENT-BASED SOFTWARE

ABUBAKAR ELSAFI ALI AHMED

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Doctor of Philosophy (Computer Science)

Faculty of Computing
Universiti Teknologi Malaysia

AUGUST 2017

To my beloved parents, brothers and sisters who always support and encourage me in the good times as well as the bad times

ACKNOWLEDGEMENT

First and foremost, my praises and thanks to Almighty Allah (S.W.T), the Most Gracious the Most Merciful, who gave me the health, strength, knowledge, encouragement and patience to accomplish this research. May the peace and blessings of Allah be upon our Prophet Mohammed (S.A.W).

I wish to express my deep gratitude to my supervisor Assoc. Prof. Dr. Dayang Norhayati binti Abang Jawawi, Deputy Dean (Academic), Faculty of Computing, Universiti Teknologi Malaysia (UTM) for her guidance to the right way and continuous support which helped me to stay focused from the beginning to the end of the research process. I would like to thank her because she has never been too busy to keep an eye on my progress in spite of her numerous obligations. She has greatly helped me in a lot of ways throughout this study. Again, I owe her my deepest thanks.

In addition, I am grateful to my colleagues at Embedded & Real-Time Software Engineering Laboratory (EReTSEL) for the useful discussions, guidance, advice and knowledge sharing in this field. Without their contribution, interest and guidance I would not be able to complete this thesis.

I would like to mention my beloved parents, brothers and sisters who has always believed in me and had always supported me during my stay in Malaysia and eagerly await my arrival back home. Their prayers have always been a source of inspiration and encouragement for me. I thank you all from the core of my heart because your prayers and well wishes made it possible for me.

There is a common Arabic saying: "I am indebted forever to whoever teaches me something, even if it is a single letter", so my thanks are due to everyone who taught me something during my PhD study. Finally, my appreciation also goes to all my friends here at UTM.

ABSTRACT

Many issues and challenges could be identified when considering integration testing of Component-Based Software Systems (CBSS). Consequently, several research have appeared in the literature, aimed at facilitating the integration testing of CBSS. Unfortunately, they suffer from a number of drawbacks and limitations such as difficulty of understanding and describing the behavior of integrated components, lack of effective formalism for test information, difficulty of analyzing and validating the integrated components, and exposing the components implementation by providing semi-formal models. Hence, these problems have made it ineffective to test today's modern complex CBSS. To address these problems, a model-based approach such as Model-Based Testing (MBT) tends to be a suitable mechanism and could be a potential solution to be applied in the context of integration testing of CBSS. Accordingly, this thesis presents a model-based integration testing technique for CBSS. Firstly, a method to extract the formal finite state behavioral models of integrated software components using Mealy machine models was developed. The extracted formal models were used to detect faulty interactions (integration bugs) or compositional problems between integrated components in the system. Based on the experimental results, the proposed method had significant impact in reducing the number of output queries required to extract the formal models of integrated software components and its performance was 50% better compared to the existing methods. Secondly, based on the extracted formal models, an effective model-based integration testing technique (MITT) for CBSS was developed. Finally, the effectiveness of the MITT was demonstrated by employing it in the air gourmet and elevator case studies, using three evaluation parameters. The experimental results showed that the MITT was effective and outperformed Shahbaz technique on the air gourmet and elevator case studies. In terms of learned components for air gourmet and elevator case studies respectively, the MITT results were better by 98.14% and 100%, output queries based on performance were 42.13% and 25.01%, and error detection capabilities were 70.62% and 75% for each of the case study.

ABSTRAK

Pelbagai isu dan cabaran dapat dikenal pasti apabila mempertimbangkan ujian integrasi bagi Perisian Sistem Berasaskan Komponen (CBSS). Oleh yang demikian, beberapa penyelidikan telah dilaksanakan dalam kajian lepas yang bertujuan untuk memudahkan ujian integrasi bagi CBSS. Namun, kajian tersebut mengalami beberapa kelemahan dan batasan seperti kesukaran dalam memahami dan menggambarkan tingkah laku komponen bersepadu, kekurangan formalisme yang berkesan bagi ujian maklumat, kesukaran dalam menganalisis dan mengesahkan komponen bersepadu, dan pendedahan pelaksanaan komponen dengan menyediakan model separa formal. Oleh itu, masalah ini telah membuatnya tidak berkesan untuk menguji kompleks CBSS moden pada hari ini. Bagi menangani masalah tersebut, pendekatan berasaskan model seperti Ujian Berasaskan Model (MBT) cenderung menjadi mekanisme yang sesuai dan boleh menjadi penyelesaian yang berpotensi untuk digunakan dalam konteks ujian integrasi CBSS. Sehubungan itu, kajian ini membentangkan teknik ujian integrasi berasaskan model untuk CBSS. Pertama, satu kaedah untuk mengekstrak model formal tingkah laku keadaan terhingga bagi integrasi komponen perisian bersepadu menggunakan model mesin Mealy telah dibangunkan. Model formal yang diekstrak digunakan untuk mengesan interaksi yang tidak berfungsi (kesilapan integrasi) atau masalah komposisi antara komponen bersepadu dalam sistem. Berdasarkan keputusan kajian, kaedah yang dicadangkan mempunyai kesan yang ketara dalam mengurangkan bilangan pertanyaan keluaran yang diperlukan untuk mengekstrak model rasmi komponen perisian bersepadu dan prestasinya adalah 50% lebih baik berbanding dengan kaedah yang sedia ada. Kedua, berdasarkan model formal yang telah diekstrak, teknik ujian integrasi berasaskan model (MITT) untuk CBSS telah dibangunkan. Akhirnya, keberkesanan MITT ditunjukkan dengan menggunakannya dalam kajian tempahan makanan dalam penerbangan dan kajian kes lif, menggunakan tiga parameter penilaian. Keputusan kajian menunjukkan bahawa MITT berkesan dan mengatasi teknik Shahbaz dalam kajian tempahan makanan dalam penerbangan dan kajian kes lif. Dari segi komponen yang dikaji untuk kajian kes tempahan makanan dalam penerbangan dan kajian kes lif, masing-masing, keputusan MITT adalah lebih baik prestasinya sebanyak 98.14% dan 100%, permintaan keluaran berdasarkan prestasi adalah 42.13% dan 25.01%, dan keupayaan pengesanan ralat adalah 70.62% dan 75% bagi setiap kajian kes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BIT	–	Built-in Testing
CBSD	–	Component-Based Software Development
CBSE	–	Component-Based Software Engineering
CE	–	Counterexamples
CBS	–	Component-Based Software
CBSS	–	Component-Based Software Systems
CD	–	Component Developer
CDT	–	Component Deployment Testing
COTS	–	Components-Off-The-Shelf
CUL	–	Component Under Learning
CU	–	Component User
C+ BIT	–	Component+ Built-in Testing
CIG	–	Component Interaction Graph
CWB	–	Edinburgh Concurrency Workbench
DPE	–	Date Parsing Exception
DFA	–	Deterministic Finite-state Automata
EFSM	–	Extended Finite State Machine
ERTS	–	Embedded Real-Time Systems
FSM	–	Finite State Machine
GOSD	–	Graph of Sequence Diagram
HVAC	–	Heating, Ventilation, & Air Conditioning
IAE	–	Illegal Input Exception
IIE	–	Invalid Input Exception
IT	–	Independent Tester

LTS	–	Labeled Transition Systems
MBT	–	Model-Based Testing
MBIT	–	Model-Based Integration Testing
NFA	–	Non-deterministic Finite Automata
NFE	–	Number Format Exception
NPE	–	Null Pointer Exception
OCL	–	Object Constraint Language
ONFSM	–	Observable Non-deterministic Finite State Machines
PFSM	–	Parameterized Finite State Machines
RF	–	Reduction Factor
RALT	–	Rich Automata Learning and Testing
SCT	–	Software Component Testing
STECC	–	Self-TEsting COTS Components
SUL	–	System Under Learning
SUT	–	System Under Test
TFM	–	Transaction Flow Model
UML	–	Unified Modeling Language
UTP	–	UML Testing Profile
UTL	–	Unable To Launch

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

Since the early 1990's, software development community was encountering various challenges and issues in developing software applications (Sneed, 2010). Today's modern software applications have become increasingly larger in scale and inherited several complex characteristics (Chakraborty and Chaki, 2016; Ghazi *et al.*, 2015; Lity *et al.*, 2015). Current software applications are composed of several sub-systems, more complicated, evolve over time into different versions, exist in many different variants, distributed amongst network and very critical (Di Ruscio *et al.*, 2014; Lochau *et al.*, 2014; Tran *et al.*, 2015). Consequently, this complex characteristic makes the development of modern software systems very expensive (Guan and Offutt, 2015). At the same time, software production cost and time-to-market for developing and delivering software applications need to be reduced due to the customer demands and the current competition amongst software businesses or companies to meet market demands (Kajtazovic *et al.*, 2014; Muschevici *et al.*, 2015). Hence, achieving the goals of on-time delivery and quality becomes more challenging (Alégroth *et al.*, 2015; Kaur and Batolar, 2015; Meyerer and Hummel, 2014). On the other hand, as today's software applications become more and more complex over time in terms of size, effort and cost, software quality accordingly have become increasingly important (Elhag *et al.*, 2013; Farjaminejad *et al.*, 2014; Goeb and Lochmann, 2011). Given these reasons, to cope with the challenges that are being faced by software community, software engineers and developers try to look for innovative alternative approaches that facilitate the development of current complex and very critical software applications (Bui, 2005; Mahmood *et al.*, 2015; Patel *et al.*, 2012).

Consequently, software engineers and developers found that today's large, complex, high quality software applications could be efficiently developed partially

if not completely, by reusing and integrating pre-built, pre-tested, high-quality, well-defined independent and "plug and play" sub-systems denoted by "software components" that their operations have been previously tested as a part of successful applications (Kaliraj *et al.*, 2014; Siddiqui and Tyagi, 2016). This idea gave birth to a very cost-effective, attractive, fast and efficient research branch in the area of software engineering known as Component-Based Software Engineering (CBSE) (Le and Pham, 2012; Tarawneh *et al.*, 2011). Therefore, building a software applications from prefabricated small pieces of software parts or components is seen as a solution to these problems (Dimri, 2015; Kaur and Tomar, 2015). As a consequence, Component-Based Software Systems (often referred as CBSS or CBS) have become a heart of most modern software applications. It has been a common trend in system development nowadays and the key benefits introduced by the CBSE approach (Pramsohler *et al.*, 2015; Shu-Fen *et al.*, 2010). Hence, CBSE paradigm has introduced significant changes in the development of current modern complex software applications thanks to their solid advantages (Tiwari and Chakraborty, 2015; Tran *et al.*, 2015).

CBSE approach has a great impact in the last few years in a wide variety of application areas such as business applications, automotive and telecommunications, distributed control applications, web-based applications, scientific applications, medical and healthcare applications, and others, including several types of Embedded Real-Time Systems (ERTS) (Guan and Offutt, 2015; Orso and Rothermel, 2014; Tao *et al.*, 2015; Zaki *et al.*, 2015). These current software systems require to ensure high degree of quality, reliability and security as well as safety. Therefore, any bug and error in these applications can affect and do the serious damage to the economies, businesses, environment as well as loss of lives. Hence, testing becomes one of the important activities and a fundamental task of software engineering is linked with the development effort of any software application with the purpose of finding faults (Ahmed and Ibrahim, 2015; Elghondakly *et al.*, 2016; Lachmann *et al.*, 2015). Unfortunately, testing today's software applications in general is an expensive activity in software development life cycle in terms of time and budget as well as other resources (Bertolino, 2007; Mahmood, 2011).

Literature suggests that 30% to 60% of the development time of a software product is dedicated to testing (Afzal *et al.*, 2016; Brar and Kaur, 2015; Ellims *et al.*, 2006; Harman *et al.*, 2015; Mohi-Aldeen *et al.*, 2017). In spite of that, it is a widely used methodology and essential stage used to evaluate the functionality of software applications, increases the confidence of the developers in the reliability and correctness of software when it will be released (Moiz, 2017). It also improves quality

of operation during deployment environment, by revealing errors and failures in order to produce high-quality software applications, especially for systems being developed by integrating prefabricated “plug and play” subsystems or software components.

Nowadays, integration of software components is a common trend and major technique of modern software development (Castro and Francisco, 2013; Groz *et al.*, 2015). Therefore, in CBSE several components, often pre-built and third-party components known as Components-Off-The-Shelf (COTS) coming from outside have to be integrated together in order to build a CBSS, instead of developing the systems from scratch (Andreou and Papatheocharous, 2016; Groz *et al.*, 2008; Shahbaz and Groz, 2014; Verma, 2012). In spite of the reality that a software component might go through different levels of testing, unit testing still cannot ensure the behavior and reliability of software component after integration in a new environment (Brohi and Jabeen, 2012; Gupta, 2015; Mahmood *et al.*, 2007). Therefore, the system developers should check to ensure that the components developed separately should work properly when integrated. Furthermore, to guarantee the correct functionality of the system, a wide number of possible interactions between integrated software components in the system may need to be tested (Guan and Offutt, 2015). Additionally, many faults may not be obvious until integration and some complex behaviors can only be observed when related components are integrated (Holling *et al.*, 2016). Thus, testing each component independently does not eliminate the need for integration testing. Therefore, integration testing, which bridges component unit testing and component system testing, plays an important role in the testing process of Component-Based Software Development (CBSD) life cycle (Sirohi and Parashar, 2013).

In integration testing phase, the individual software components are assembled and verified as a group to attain a high level of quality and reliability (Belli *et al.*, 2009; Khan and Nadeem, 2013). Moreover, integration testing is essential level of quality assurance that minimizes the risk of the system not working effectively and efficiently, and focuses on the prevention of integration bugs (Khan and Singh, 2012). Therefore, it plays a substantial part in detecting faults during a CBSD life cycle (Mahmood, 2011). Approximately, 40% of the software errors are discovered and revealed during integration testing (Kaur *et al.*, 2011). On the other hand, integration testing of CBSS is most time consuming and expensive part of the testing process in CBSD, and received significantly little attention from practitioners and researchers in the field of software engineering (Bai *et al.*, 2001; Gao *et al.*, 2003; Ning *et al.*, 2013; Shashank *et al.*, 2010; Shukla and Marwala, 2012).

1.2 Research Motivation

When considering integration testing of CBSS; many issues and challenges could be identified specially when considering integrated black-box software components (Bertolino and Polini, 2003; Beydeda and Gruhn, 2003b; Khan *et al.*, 2011; Machado *et al.*, 2007; Reza and Cheng, 2012). In most cases, the source code, detailed documentations, design specifications, and formal models of integrated components are missing or not available to the system designers, making system integration and integration testing are very time consuming and more quite challenging (Haser, 2015). The consequences of these challenges, several works have been proposed during the last decade, aiming to facilitate integration testing of CBSS. Here in the following sections the issues and challenges that are related to the scope of this thesis directly or indirectly are summarized, and the efforts made by researchers to tackle it.

The first challenge deals with a lack of information exchanged between component producers and component consumers. Component developers and component users need to exchange different types of information during the development of the component itself and during the development of CBSS in order to facilitates the testing process, and to develop high-quality and reliabe CBSS. Unfortunately, unavailable information limits the capacity of both component developers and component users or even the independent tester (third-party tester) to test candidate components efficiently. Therefore, there is an effort made by several researchers to aggregate valuable information to the component in order to facilitate the integration testing activities at the component developers' side and to minimize as much as possible the dependence of component users and the independent tester (third-party tester) on the information provided by the component providers.

In 1998, Liu and Richardson (1998) proposed a technique to capture information related to the usage of the component inside source code snippets. In 1999, Harrold *et al.* (1999) initiated the idea of metadata by proposing a technique for analyzing and testing CBSS from two perspectives of component providers and component users. In 2001, Orso *et al.* (2001) suggested another category of summary information to support component testing called component metadata. Component users can then use the information provided by the metadata to test and analyze a CBSS. Edwards (2001) proposed the reflective wrapper technique that analyzes the best way to package specification and verification information of formally specified components for distribution to component consumers. In 2003, Bertolino and Polini

(2003) proposed component deployment technique for component users to facilitate the testing within a user's target environment of a component independently developed. Silva *et al.* (2009) presents a technique covered by a CASE tool integrated in the development environment to support components integration testing aiming to reduce the lack of information between component producers and component consumers. In 2010, Naseer *et al.* (2010) presented a technique to use metadata technique for CBS black box testing and developed a tool which takes <.dll> component. Brohi and Jabeen (2012) proposed a technique that enhances component testability and to facilities the integration testing by defining a uniform information flow in the component life cycle.

Indeed, all the proposed previous related research that follows in this category, however, don't handle heterogeneous components and have an effect on the implementation transparency of the component, which is an important characteristic in software component.

Unavailability of the components source code, or internal workings (structure) of the integrated components might not be accessed by a component users and the independent tester (third-party tester) due to organizational or legal restriction, will affect and reduce the controllability and testability of candidate components, and hence the overall integration testing of CBSS. Consequently, a number of proposed solutions provided by several researchers to equip a software components with a specific testable architecture that permits component users and the independent tester (third-party tester) to execute test cases easily in order to support self-testing and to facilitate the integration testing process.

In 1999, Wang *et al.* (1999) proposed the idea of Built-in Testing approach (BIT) to increase component testability. BIT based on the development of test cases in component source code as additional member functions with the normal member functions. Consequently, Atkinson and Grob (2002) presented an example of BIT technique, called component+. It is related to contract testing in model-driven component-based development. In the same year, Gao *et al.* (2002) proposed the testable beans technique. The idea of BIT was extended by Mahmood (2011) to present a component-based software integration testing technique to prioritize test cases and identify integration test criteria using software complexity measures.

In 2001, Martins *et al.* (2001) has presented a technique for Self-testable software component. The proposed technique is another example to add extra

information to software component intentionally to improve component testability by integrating testing resources into it. Concat tool was developed to support the proposed technique. In 2003, Beydeda and Gruhn (2003a) presented the Self-Testing COTS Components (STECC) technique that augment components with functionality specific to testing tools that is capable of conducting some or all activities of the component user's testing processes.

Accordingly, due to the unavailability of integrated software component source code, and with the lack of component information, detailed documentations, complete functional behavior descriptions and design specifications for analysis and testing the systems of integrated components, specifically when one considers black-box software components, existing integration testing techniques under this category has become ineffective to handle and detect erroneous interactions of components during the integration testing phase, and fail to test today's modern complex CBSS. Additionally, in most cases the formal models for analysis and testing the integrated components, and which will help to understand their possible behaviors in the system are always missing or not available to the system designers. Furthermore, there could be a large number of possible interactions between integrated software components which are undesirable and which could affect the function of each others, hence, these interactions may need to be tested to ensure the correct functionality of the CBSS. As a result, some complex behaviors are not observed until related components are integrated and many faults may not be visible until integration, making integration testing very quite challenging by the existing integration testing techniques and most time consuming task. Moreover, this invisible behavior of a component can affect the behavior of the overall CBSS.

The Unified Modeling Language (UML) and its diagrams are commonly used and gained wide acceptance amongst researchers to address the necessity for additional information by appending some UML models with software components. Accordingly, the techniques consist of UML models such as (Barisas *et al.*, 2013; Gallagher *et al.*, 2006; Hartmann *et al.*, 2000; Kaur *et al.*, 2011; Machado *et al.*, 2007; Mussa and Khendek, 2012; Shang and Zhang, 2006; Wu *et al.*, 2003; Zheng and Bundell, 2008, 2007a,b) have been proposed in order to facilitate the integration testing process at the component user side.

However, the drawbacks of UML based integration testing approach is that, with the help of reverse engineering its possible to modify a component and to access the component source code by using some UML reverse engineering tools. Furthermore, these UML models is unrealistic because COTS evolve along

time to incorporate additional requirements, which quickly invalidates the original project (Castro and Francisco, 2013). Moreover, the techniques consist of semi-formal models such as UML models affecting the implementation transparency of software components. Additionally, these semi-formal models is not sufficient for the component users to understand its behavior completely, and it cannot be used as input to the existing MBT approaches. Hence, the traditional techniques for MBT approaches such as static analysis, program slicing, invariant detection, model extraction, validation and verification also has become ineffective when one considers systems of integrated black-box software components.

To conclude, despite several existing proposals, the existing works discussed in this section of adding additional structure for reliable use of component applications, and/or adding information with software components to facilitate the integration testing process suffers from a number of drawbacks and limitations, leading to ineffective testing and, ultimately, to poor software quality. These drawbacks and limitations are very important and should be resolved. Hence, new solutions have to be developed to cope with these drawbacks and limitations. The drawbacks and limitations of the existing works that are directly related to the scope of this thesis are highlighted here:

i. Difficulty of understanding and describing the behaviors of integrated components, due to the frequent lack of information and/or implementation details. Moreover, the formal models of integrated components that can be used for analysis, testing and documentation, and which will help to understand their possible behaviors in the system are always missing or not available to the system designers.

ii. It is impossible to use it in some cases, for instance, when there is no formal model to understand the possible behavior of the integrated components. A key problem, however, is the construction of models that describe the intended behavior of the integrated components in a system.

iii. Difficulty of analyzing and validating the integrated components, due to the restricted access of components source code, detailed documentations is not sufficient to solve details about its interaction with other components, absence of components models to check the possible interaction between the components, and other development information to the system designers.

iv. Lack of accurate and effective formalism for information representation (internal structure of a component is generally unknown). Hence, how to query and retrieve test information effectively has become the key problem while reusing test information. Due to that, the interpretation of information is not clear due to it is non-uniformity, hence requires understanding the representation prior to interpreting the meaning.

v. Some techniques expose component implementation by providing semi-formal models such as metadata, BIT, and UML models, allowing reverse engineering to access the source code of software component, thus affecting the implementation transparency of software components. Furthermore, the implementation transparency property raises some difficulties when the CBSS is to be tested. Moreover, these informal models cannot be used as input to the existing Model-Based Testing (MBT) approaches, which is a new technique which rely on explicit or accurate models for testing purposes, and aims to make testing more effective and more efficient.

In view of the above background, and in order to address the important challenges and limitations of the existing proposals, this thesis aimed at proposing an integration testing technique for CBSS, by exploits the use of learning and testing approach for integration testing of CBSS. Therefore, the proposed technique in this thesis combining model learning and testing techniques for testing of a system of integrated software components.

1.3 Statements of the Problem

Engineering high-quality CBSS is essential for the involved enterprises and demands interest from both academia and industry. With the increasing complexity of today's modern CBSS, verification and validation techniques are becoming more and more important. Therefore, integration testing has become a very essential activity linked with the development effort of any software applications and most important means to ensure the quality of today's modern CBSS. Unfortunately, nowadays many difficulties arise in integration testing of CBSS leads to new challenges, and it has become a more and more complex task that significantly influenced by a number of factors. Even so, integration testing of CBSS has been one of the open research area that is rarely investigated and received significantly little attention from practitioners and researchers in the field of software engineering, and remains comparatively

less well-solved. Therefore, it requires the support of tools, methodologies and well established techniques to mitigate the challenges and limitations of the existing works, and to support the development of high-quality CBSS. On the other hand, understanding the behavior and testing the integrated software components is another challenging task, due to the unavailability of their source code, updated specifications or formal models. Moreover, obtaining the accurate formal models for existing software components, which precisely describe the behavior of the integrated software components is still an open and interesting problem.

The recognized problems of the integration testing of CBSS are presented in Figure 1.1. However, by understanding the problem background which has been discussed in the previous section, it can be concluded that continues efforts and further works still required, and a new solution for improvement should be developed to mitigate the above limitations and gaps left by past research works effectively in order to enhance the development of today's modern CBSS.

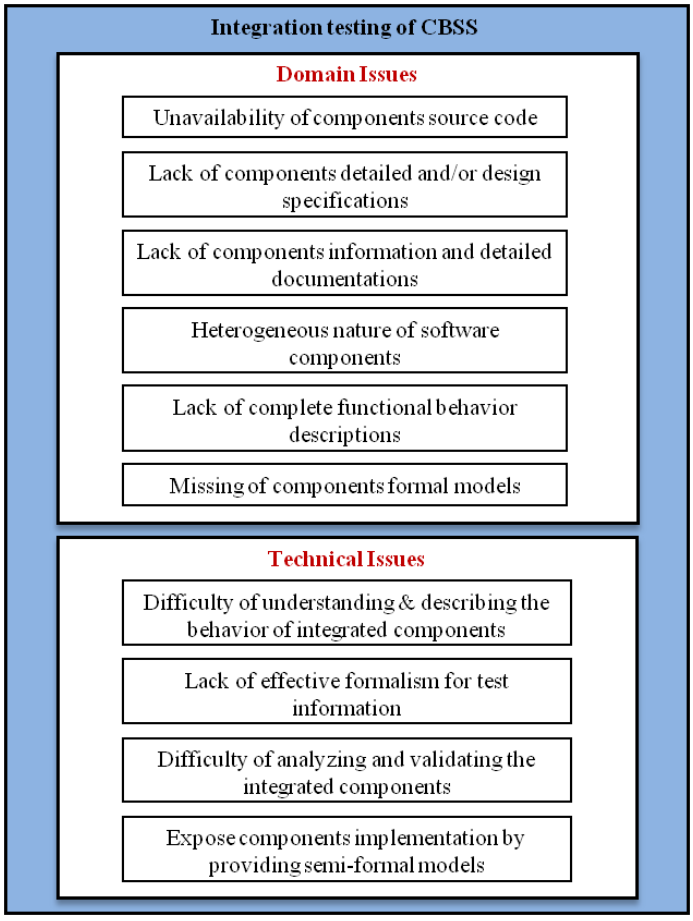

Figure 1.1: Recognized problems

The environment of a CBSS can be seen as a form of black-box (service interface). Therefore, a model-based approaches such as MBT can be seen as potential solution. Since MBT is a black-box approach, it tend itself to be suitable mechanism to deal with CBSS problems. Accordingly, among the existing testing techniques, MBT is a promising candidate to be applied in the context of integration testing of CBSS. Despite the amount of literature on integration testing, model-based integration testing techniques are quite limited (Dias Neto *et al.*, 2007). MBT has become a common trends which have added many values to the engineering of software projects. MBT has more advantages, and can well support component integration testing. Besides the automatic test case generation, another relevant characteristic for testing components is the adoption of formal models and black-box testing strategy (Haser, 2015). Black-box testing are appropriate for integration testing of CBSS because internal structures of integrated software components are always missing or not available to the system designers, the complexity of interactions and test harness can be abstracted, the formality of the model contribute to more reliable tests. Furthermore, several benefits such as (high level of automation, reducing cost and time for testing, high fault detection rate and generating tests automatically) were obtained from the adequate application of MBT to software systems.

1.4 Research Questions

This research intends to propose an integration testing technique for CBSS in order to mitigate above mentioned problems. Despite the powerful features of MBT approach, most existing techniques provide only limited support for integration testing. This leads to the main research question to be answered in this study:

"How is it possible to use MBT to develop an effective model-based integration testing technique for CBSS?"

To address the primary research question given above, it is further broken down into sub-questions.

RQ1: How to understand the behavior of integrated software components in the system?

RQ2: How to develop an effective integration testing technique for CBSS in order to overcome the identified challenges?

RQ3: How to measure the effectiveness of the proposed technique?

1.5 Research Goal

This research work concentrates on the problem of testing the integrated software components in the system in the missing of their formal behavioral models. Given a set of black-box software components that are integrated in a system, the first major goal of this study is to infer/extract the formal models which describes the behaviors of integrated software components, by proposing a method to infer the approximated finite state behavioral models in term of finite state machine (Mealy machines), that represents the precise description of the intended behaviors of the integrated components formally directly from the components, using the idea of active learning approach. Then, the second major goal of the study is to propose an effective model-based integration testing technique, which combines model learning and testing techniques, to identify the faulty interaction between the integrated components in a system based on the extracted formal models and with the help of learning and testing approach.

1.6 Research Objectives

To achieve the goal, the following four objectives need to be undertaken with the aim of finding answers to the research questions:

i. To propose a method that extracts the formal finite state behavioral models of integrated software components using active learning approach and benchmark the performance based on the number of output queries.

ii. To develop an effective integration testing technique for CBSS using the extracted models and with the help of learning and testing approach.

iii. To demonstrate and measure the effectiveness of the proposed technique by applying the proposed technique in the selected applications as case studies, using three evaluation parameters, namely learned components, output queries, and error detection capability.

iv. To evaluate and compare the proposed technique against Shahbaz technique based on learned components, output queries, and error detection capability.

1.7 Significance of the Study

The significance of this study is to mitigate the limitations and gaps left by past research works in integration testing of CBSS, by proposing an effective integration testing technique of CBSS, which combines model learning and testing techniques for testing of a system of integrated black-box software components. Thus, this thesis exploits the use of learning and testing approach for integration testing of CBSS. The study proposes solutions into two directions:

i. **Reverse engineering:** Understanding the behaviors of the integrated black-box software components, by deriving (extracting) the formal models of the components. In this study, the software components are learned in order to extract their formal finite state behavioral models as Mealy machine models.

ii. **Validation:** Developing an effective model-based technique for integration testing of CBSS. Thus, the integrated software components is tested and analyzed using their learned formal models (Mealy machine models). In this study, the use of components formal models will help in revealing compositional problems or faulty interactions (integration bugs) between integrated software components and general errors in the system.

1.8 Scope of the Study

The scope of this research work has been limited to the following aspects:

i. **Integration testing:** As discussed before in this chapter, in the CBSD life cycle, three basic kinds of testing are needed in order to detect and reveal errors, namely "unit testing (component testing), integration testing (deployment testing), and finally system testing". A brief description of this three levels of testing will be introduced in the next chapter. However, this research is concerned only on integration testing in the context of CBSS, and does not cover the other testing levels. Thus, the issues related to other testing levels are not dealt with in this study.

ii. **Components are black boxes:** The integrated components in the system may have different levels of exposure depending upon how much information about them is available. In literature, the terms black box, gray box and white box are used with reference to different levels of closure of the component internal

essence. This study considers that all components are black boxes, i.e., their functional specifications and implementation details are not available. However, the basic set of input symbols that can be given to a component through it is input interfaces are known, and for each input, the corresponding output of the component can be observed through it is output interfaces.

iii. **Modeling level:** The component exhibits regular behaviors, i.e., the component can be modeled as a finite state machine (Mealy machine). This study intends to learn only the behaviors prescribed by the control structure of the finite state model. Moreover, the study do not assume to know the upper bound on the number of states in the components. Instead of hunting for exact learning, the study aims to learn approximate models that are expressive enough to provide powerful guidance for testing and to enhance the behavior understanding of the integrated software components, and thus, of the system.

iv. **Focus on functional aspects:** This study focuses on behavior learning and studying the interactions between the integrated components and their functional aspects in the system. Therefore, the study are not dealing with other details, for instance, security, timing, and performance issues in the system.

v. **Case studies and their assumptions:** The proposed research work in this thesis has been validated using four different case studies that large enough to get some interesting results. Therefore, different case studies from different domains have been used in this research. Furthermore, in order to check the effectiveness of the proposed technique and to compare its results with the existing proposal's results, the selected case studies are fully developed based on CBSD.

vi. **Benchmarking with existing techniques:** To the best of our knowledge, Shahbaz and Groz (2014) is the only work found in the literature that uses learning and testing approach for CBSS. Therefore, the technique proposed by Shahbaz and Groz (2014) is the best and closest work to compare its experimental results with the obtained experimental results of the proposed technique in this thesis.

1.9 Thesis Outline

This thesis is organized in seven chapters. The structure of these chapters as follows:

Chapter 2, *Literature Review*. This chapter is associated with literature review. First, the chapter provides information about testing CBSS. Then, a comprehensive review of integration testing of CBSS will be provided in details. Next, the chapter presents a discussion on the related work on model-based integration testing. Moreover, a detailed description of learning and testing approach will be provided. Next, an overview of the most important active learning tools will be given. Learning finite state machine will be highlighted also in this chapter.

Chapter 3, *Research Methodology*. This chapter explains comprehensively the research methodology used in this thesis in order to show how the objectives are achieved. This includes the research process and activities involved in each phase to depicts the flow of research step by step, and the research framework to show the main parts and components of the research. At last, the description and comparison of case studies that will be used in this research will be provided in this chapter.

Chapter 4, *A Method to Extracting the Formal Finite State Behavioral Model*. This chapter describes in details the proposed finite state behavioral model extraction method. In addition, this chapter also explains and discusses the results related to the experimental evaluation of the proposed method using case studies.

Chapter 5, *The Proposed Model-based Integration Testing Technique*. In this chapter, an integration testing technique for testing the integrated black-box software components in a system using approximated (partial) models of software components is proposed. Precisely, this chapter presents the structure and development steps of the proposed technique. The discussion includes all the necessary elements that related to the proposed technique. An illustrative example will be used to clarify the implementation of the proposed technique.

Chapter 6, *Evaluation and Comparison of MITT*. This chapter discusses the results of evaluating and comparing the proposed technique with other current techniques in order to explain the strengths and weaknesses related to the proposed technique. Therefore, the chapter explains the evaluation of the proposed technique in details. In particular, the experimental results are provided and discussed in details.

Chapter 7, *Conclusion and Future Work*. This chapter concludes the thesis by highlighting the summary, achievements of research objectives covered in this thesis, the contributions of research, and finally, the future work are elaborated.

REFERENCES

Abel, A. and Reineke, J. (2016). Gray-Box Learning of Serial Compositions of Mealy Machines. In Rayadurgam, S. and Tkachuk, O. (Eds.) *NASA Formal Methods: 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings*. (pp. 272–287). Cham: Springer International Publishing.

Afzal, W., Alone, S., Glocksien, K. and Torkar, R. (2016). Software test process improvement approaches: A systematic literature review and an industrial case study. *Journal of Systems and Software*. 111, 1–33.

Ahmed, M. and Ibrahim, R. (2015). A Comparative Study of Web Application Testing and Mobile Application Testing. In Sulaiman, A. H., Othman, A. M., Othman, I. M. F., Rahim, A. Y. and Pee, C. N. (Eds.) *Advanced Computer and Communication Engineering Technology: Proceedings of the 1st International Conference on Communication and Computer Engineering*. (pp. 491–500). Cham: Springer International Publishing.

Aichernig, B. K. and Tappler, M. (2017). Learning from Faults: Mutation Testing in Active Automata Learning. In Barrett, C., Davies, M. and Kahsai, T. (Eds.) *NASA Formal Methods: 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*. (pp. 19–34). Cham: Springer International Publishing.

Albeladi, K. S. and Qureshi, M. R. J. (2013). Improvement of Component Integration Testing Technique. *International Journal of Information Technology and Computer Science (IJITCS)*. 5(8), 109–122.

Alégroth, E., Gao, Z., Oliveira, R. and Memon, A. (2015). Conceptualization and evaluation of component-based testing unified with visual gui testing: an empirical study. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. 13-17 April 2015. Graz, Austria: IEEE, 1–10.

Alsaeedi, A. (2016). *Improving Software Model Inference by Combining State Merging and Markov Models*. Ph.D. Thesis. University of Sheffield.

Alvaro, A., de Almeida, E. S. and de Lemos Meira, S. R. (2005). Software component certification: a survey. In *Software Engineering and Advanced Applications, 2005.*

31st EUROMICRO Conference on. 30 Aug. - 3 Sept. 2005. Porto, Portugal: IEEE, 106–113.

Ammann, P. and Offutt, J. (2008). *Introduction to software testing*. Cambridge, United Kingdom: Cambridge University Press.

Andreou, A. S. and Papatheocharous, E. (2016). Towards a CBSE Framework for Enhancing Software Reuse: Matching Component Properties Using Semi-formal Specifications and Ontologies. In Maciaszek, A. L. and Filipe, J. (Eds.) *Evaluation of Novel Approaches to Software Engineering: 10th International Conference, ENASE 2015, Barcelona, Spain, April 29-30, 2015, Revised Selected Papers*. (pp. 98–121). Cham: Springer International Publishing.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and computation*. 75(2), 87–106.

Atkinson, C. and Grob, H.-G. (2002). Built-in contract testing in model-driven, component-based development. In *Proceedings of the Workshop on Component-Based Development Processes*. April 2002. Austin, Texas USA, 1–15.

Bai, X., Tsai, W., Paul, R., Shen, T. and Li, B. (2001). Distributed end-to-end testing management. In *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*. 4-7 Sept. 2001. Seattle, Washington USA: IEEE, 140–151.

Barisas, D., Bareiša, E. and Packevičius, Š. (2013). Automated method for software integration testing based on UML behavioral models. In Skersys, T., Butleris, R. and Butkiene, R. (Eds.) *Information and Software Technologies: 19th International Conference, ICIST 2013, Kaunas, Lithuania, October 2013. Proceedings*. (pp. 272–284). Berlin, Heidelberg: Springer Berlin Heidelberg.

Basanieri, F., Bertolino, A. and Marchetti, E. (2002). The cow_suite approach to planning and deriving test suites in UML projects. In Jézéquel, J.-M., Hussmann, H. and Cook, S. (Eds.) *UML 2002 — The Unified Modeling Language: Model Engineering, Concepts, and Tools 5th International Conference Dresden, Germany, September 30 – October 4, 2002 Proceedings*. (pp. 383–397). Berlin, Heidelberg: Springer Berlin Heidelberg.

Belli, F. and Budnik, C. J. (2005). Towards self-testing of component-based software. In *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*. 26-28 July 2005. Hong Kong, China: IEEE, 205–210.

Belli, F., Hollmann, A. and Padberg, S. (2009). Communication Sequence Graphs for Mutation-Oriented Integration Testing. In *Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009. Third IEEE International Conference*

on. 8-10 July 2009. Shanghai, China: IEEE, 387–392.

Berg, T., Jonsson, B., Leucker, M. and Saksena, M. (2005). Insights to Angluin's learning. *Electronic Notes in Theoretical Computer Science*. 118, 3–18.

Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. In *Proceedings of the on Future of Software Engineering (FOSE '07)*. 23-25 May 2007. Washington, DC, USA: IEEE, 85–103.

Bertolino, A., Marchetti, E. and Polini, A. (2003). Integration of "Components" to test software components. *Electronic Notes in Theoretical Computer Science*. 82(6), 44–54.

Bertolino, A. and Polini, A. (2003). A framework for component deployment testing. In *Software Engineering, 2003. Proceedings. 25th International Conference on*. 3-10 May 2003. Portland, Oregon USA: IEEE, 221–231.

Beydeda, S. and Gruhn, V. (2003a). Merging components and testing tools: the self-testing COTS components (STECC) strategy. In *Euromicro Conference, 2003. Proceedings. 29th*. 1-6 Sept. 2003. Antalya, Turkey: IEEE, 107–114.

Beydeda, S. and Gruhn, V. (2003b). State of the art in testing components. In *Quality Software, 2003. Proceedings. Third International Conference on*. 6-7 Nov. 2003. IEEE, 146–153.

Bollig, B., Katoen, J.-P., Kern, C., Leucker, M., Neider, D. and Piegdon, D. R. (2010). libalf: The automata learning framework. In Touili, T., Cook, B. and Jackson, P. (Eds.) *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 360–364.

Brar, H. K. and Kaur, P. J. (2015). Differentiating Integration Testing and unit testing. In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*. 11-13 March 2015. New Delhi, India: IEEE, 796 – 798.

Brohi, M. N. and Jabeen, F. (2012). A Metadata-based Framework for Object-Oriented Component Testing. *International Journal of Computer Applications*. 41(15), 8–18.

Bui, B. (2005). *An interface-based testing technique for component-based software systems*. Master's Thesis. San Jose State University.

Cassel, S., Howar, F. and Jonsson, B. (2015). RALib: A LearnLib extension for inferring EFSMs. In *International Workshop on Design and Implementation of Formal Tools and Systems (DIFTS)*. 26-27 Sept. 2015. Austin, Texas USA, 1–8.

Cassel, S., Howar, F., Jonsson, B. and Steffen, B. (2016). Active learning for extended finite state machines. *Formal Aspects of Computing*. 28(2), 233–263.

Castro, L. M. and Francisco, M. A. (2013). A language-independent approach to black-box testing using Erlang as test specification language. *Journal of Systems and Software*. 86(12), 3109–3122.

Chakraborty, M. and Chaki, N. (2016). A New Framework for Configuration Management and Compliance Checking for Component-Based Software Development. In Chaki, R., Cortesi, A., Saeed, K. and Chaki, N. (Eds.) *Advanced Computing and Systems for Security: Volume 2*. (pp. 173–188). New Delhi, India: Springer India.

Cheesman, J. and Daniels, J. (2000). *UML components: a simple process for specifying component-based software*. Addison-Wesley Longman Publishing Co., Inc.

Councill, W. T. (1999). Third-party testing and the quality of software components. *IEEE software*. 16(4), 55–57.

Czerny, M. X. (2014). *Learning-based software testing: Evaluation of Angluin's L* algorithm and adaptations in practice*. Ph.D. Thesis. Karlsruhe Institute of Technology.

Damiani, F., Faitelson, D., Gladisch, C. and Tyszberowicz, S. (2016). A novel model-based testing approach for software product lines. *Software & Systems Modeling*, 1–29.

De Souza, É. F., De Santiago Júnior, V. A. and Vijaykumar, N. L. (2015). H-Switch Cover: a new test criterion to generate test case from finite state machines. *Software Quality Journal*. 24, 1–33.

Di Ruscio, D., Malavolta, I. and Pelliccione, P. (2014). The Role of Parts in the System Behaviour. In Majzik, I. and Vieira, M. (Eds.) *Software Engineering for Resilient Systems: 6th International Workshop, SERENE 2014, Budapest, Hungary, October 15-16, 2014. Proceedings*. (pp. 24–39). Cham: Springer International Publishing.

Dias Neto, A. C., Subramanyan, R., Vieira, M. and Travassos, G. H. (2007). A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*. 5 Nov. 2007. Atlanta, Georgia: ACM, 31–36.

Dimri, S. C. (2015). An Innovative Model of Software Quality Assurance for Component-Based Software Systems. *International Journal of Emerging Research in Management & Technology*. 4(7), 216–219.

Do Nascimento, N. M. and De Lucena, C. J. P. (2017). FIoT: An agent-based framework for self-adaptive and self-organizing applications based on the Internet of Things. *Information Sciences*. 378, 161–176.

Edwards, S. H. (2001). A framework for practical, automated black-box testing of component-based software. *Software Testing Verification & Reliability*. 11(2), 97–111.

El-Fakih, K., Groz, R., Irfan, M. N. and Shahbaz, M. (2010). Learning finite state models of observable nondeterministic systems in a testing context. In *22nd IFIP International Conference on Testing Software and Systems*. 8-10 Nov. 2010. Natal, Brazil, 97–102.

Elghondakly, R., Moussa, S. and Badr, N. (2016). A Comprehensive Study for Software Testing and Test Cases Generation Paradigms. In *Proceedings of the International Conference on Internet of Things and Cloud Computing (ICC '16)*. 22-23 March 2016. New York, NY, USA: ACM, 1–7.

Elhag, A. A., Elshaikh, M., Mohamed, R. and Babar, M. I. (2013). Problems and future trends of software process improvement in some Sudanese software organizations. In *Computing, Electrical and Electronics Engineering (ICCEEE), 2013 International Conference on*. 26-28 Aug. 2013. Khartoum, Sudan: IEEE, 263–268.

Ellims, M., Bridges, J. and Ince, D. C. (2006). The economics of unit testing. *Empirical Software Engineering*. 11(1), 5–31.

Farjaminejad, F., Harounabadi, A., Mirabedini, S. J. *et al.* (2014). Modeling and Evaluation of Performance and Reliability of Component-based Software Systems using Formal Models. *International Journal of Computer Applications Technology and Research*. 3(1), 73–78.

Fatima, F., Ali, S. and Ashraf, M. U. (2017). Risk Reduction Activities Identification in Software Component Integration for Component Based Software Development (CBSD). *International Journal of Modern Education and Computer Science*. 9(4), 19–31.

Gallagher, L., Offutt, J. and Cincotta, A. (2006). Integration testing of object-oriented components using finite state machines. *Software Testing, Verification and Reliability*. 16(4), 215–266.

Gao, J., Gupta, K., Gupta, S. and Shim, S. (2002). On building testable software components. In Dean, J. and Gravel, A. (Eds.) *COTS-Based Software Systems: First International Conference, ICCBSS 2002 Orlando, FL, USA, February 4–6, 2002 Proceedings*. (pp. 108–121). Berlin, Heidelberg: Springer Berlin Heidelberg.

Gao, J. Z., Tsao, J., Wu, Y. and Jacob, T. H.-S. (2003). Testing and Quality Assurance for Component-Based Software, Artech House. *Inc., Norwood, MA*.

Ghazi, A. N., Petersen, K. and Börstler, J. (2015). Heterogeneous Systems Testing Techniques: An Exploratory Survey. In Winkler, D., Biffl, S. and Bergsmann, J. (Eds.) *Software Quality. Software and Systems Quality in Distributed and Mobile Environments: 7th International Conference, SWQD 2015, Vienna, Austria, January 20-23, 2015, Proceedings*. (pp. 67–85). Cham: Springer International Publishing.

Goeb, A. and Lochmann, K. (2011). A software quality model for SOA. In *Proceedings of the 8th international workshop on Software quality*. 5-9 Sept. 2011. Szeged, Hungary: ACM, 18–25.

Groz, R., Li, K. and Petrenko, A. (2015). Integration testing of communicating systems with unknown components. *annals of telecommunications*. 30(3-4), 107–125.

Groz, R., Li, K., Petrenko, A. and Shahbaz, M. (2008). Modular system verification by inference, testing and reachability analysis. In Suzuki, K., Higashino, T., Ulrich, A. and Hasegawa, T. (Eds.) *Testing of Software and Communicating Systems: 20th IFIP TC 6/WG 6.1 International Conference, TestCom 2008 8th International Workshop, FATES 2008 Tokyo, Japan, June 10-13, 2008 Proceedings*. (pp. 216–233). Berlin, Heidelberg: Springer Berlin Heidelberg.

Groz, R., Oriat, C. and Brémond, N. (2016). Inferring Non-resettable Mealy Machines with n States. In *Proceedings of The 13th International Conference on Grammatical Inference*. October 5-7, 2016. Delft, The Netherlands, 30–41.

Guan, J. and Offutt, J. (2015). A model-based testing technique for component-based real-time embedded systems. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*. 13-17 April 2015. Graz, Austria: IEEE, 1–10.

Gupta, N. (2015). Stepping Towards Component-Based Software Testing Through A Contemporary Layout. *International Journal of Computer Science & Engineering Technology (IJCSET)*. 6(8), 504–508.

Harman, M., Jia, Y. and Zhang, Y. (2015). Achievements, open problems and challenges for search based software testing. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. 13-17 April 2015. Graz, Austria: IEEE, 1–12.

Harrold, M. J., Liang, D. and Sinha, S. (1999). An approach to analyzing and testing component-based systems. In *First International ICSE Workshop on Testing Distributed Component-Based Systems*. 16 May 1999. Los Angeles, CA: ACM, 333–347.

Hartmann, J., Imoberdorf, C. and Meisinger, M. (2000). UML-Based Integration Testing. In *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '00)*. 22-25 Aug. 2000. Portland, Oregon: ACM, 60–70.

Haser, F. (2015). Non-Intrusive Documentation-Driven Integration Testing. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. 13-17 April 2015. Graz, Austria: IEEE, 1–2.

Holling, D., Hofbauer, A., Pretschner, A. and Gemmar, M. (2016). Profiting from Unit Tests for Integration Testing. In *Software Testing, Verification and Validation (ICST), 2016 IEEE International Conference on*. 11-15 April 2016. Chicago, Illinois, USA: IEEE, 353–363.

Howar, F., Meinke, K. and Rausch, A. (2016). Learning Systems: Machine-Learning in Software Products and Learning-Based Analysis of Software Systems. In Margaria, T. and Steffen, B. (Eds.) *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II*. (pp. 651–654). Cham: Springer International Publishing.

Hungar, H., Niese, O. and Steffen, B. (2003). Domain-specific optimization in automata learning. In Hunt, W. A. and Somenzi, F. (Eds.) *Computer Aided Verification: 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003. Proceedings*. (pp. 315–327). Berlin, Heidelberg: Springer Berlin Heidelberg.

Irfan, M. N., Oriat, C. and Groz, R. (2010). Angluin style finite state machine inference with non-optimal counterexamples. In *Proceedings of the First International Workshop on Model Inference In Testing*. 12-16 July 2010. Trento, Italy: ACM, 11–19.

Isberner, M. (2015). *Foundations of active automata learning: an algorithmic perspective*. Ph.D. Thesis. Universität Dortmund.

Isberner, M., Howar, F. and Steffen, B. (2014). The TTT algorithm: A redundancy-free approach to active automata learning. In Bonakdarpour, B. and Smolka, S. A. (Eds.) *Runtime Verification: 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*. (pp. 307–322). Cham: Springer International Publishing.

Isberner, M., Howar, F. and Steffen, B. (2015). The Open-Source LearnLib. In Kroening, D. and Păsăreanu, S. C. (Eds.) *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015,*

Proceedings, Part I. (pp. 487–495). Cham: Springer International Publishing.

Jabeen, F. and Rehman, M. J. U. (2005). A framework for object oriented component testing. In *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on*. 17-18 Sept. 2005. Islamabad, Pakistan: IEEE, 451–460.

Kajtazovic, N., Preschern, C., Höller, A. and Kreiner, C. (2014). Towards pattern-based reuse in safety-critical systems. In *Proceedings of the 19th European Conference on Pattern Languages of Programs*. 9-13 July 2014. Irsee, Germany: ACM, 1–15.

Kaliraj, S., Premkumar, N. and Bharathi, A. (2014). The Novel Life Cycle Model for Component Based Software System Based on Architecture Quality Using KCW Framework. *International Journal of Information Technology and Computer Science (IJITCS)*. 6(9), 74–79.

Kaur, E. J., RupinderPal and Kaur, E. K. (2011). A Systematic Approach For UML Based Test-Generation For Integration Testing. *International Journal of Research in Engineering and Applied Sciences*. 1(1), 58–69.

Kaur, J. and Tomar, P. (2015). Multi Objective Optimization Model using Preemptive Goal Programming for Software Component Selection. *International Journal of Information Technology and Computer Science (IJITCS)*. 7(9), 31–37.

Kaur, P. and Batolar, N. (2015). A Review on Quality Assurance of Component-Based Software System. *IOSR Journal of Computer Engineering (IOSR-JCE)*. 17(3), 53–57.

Kearns, M. J. and Vazirani, U. V. (1994). *An introduction to computational learning theory*. Cambridge, Massachusetts, United States: MIT press.

Khalili, A., Narizzano, M. and Tacchella, A. (2016). Learning for Verification in Embedded Systems: A Case Study. In Adorni, G., Cagnoni, S., Gori, M. and Maratea, M. (Eds.) *AI*IA 2016 Advances in Artificial Intelligence: XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 – December 1, 2016, Proceedings*. (pp. 525–538). Cham: Springer International Publishing.

Khan, I. A. and Singh, R. (2012). Quality Assurance And Integration Testing Aspects In Web Based Applications. *International Journal of Computer Science, Engineering and Applications (IJCSEA)*. 2(3), 109–116.

Khan, S. and Nadeem, A. (2013). Automated Test Data Generation for Coupling Based Integration Testing of Object Oriented Programs Using Evolutionary Approaches. In *Information Technology: New Generations (ITNG), 2013 Tenth International Conference on*. 15-17 April 2013. Las Vegas, NV: IEEE, 369–374.

Khan, U. A., Al-Bidewi, I. and Gupta, K. (2011). Challenges in Component Based Software Engineering as the Technology of the Modern Era. *International Journal of Internet Computing (IJIC)*. 1(2), 67–72.

Lachmann, R., Lity, S., Lischke, S., Beddig, S., Schulze, S. and Schaefer, I. (2015). Delta-oriented test case prioritization for integration testing of software product lines. In *Proceedings of the 19th International Conference on Software Product Line*. 20-24 July 2015. Nashville, TN USA: ACM, 81–90.

Le, B. C. and Pham, N. H. (2012). A Method for Generating Models of Black-Box Components. In *Knowledge and Systems Engineering (KSE), 2012 Fourth International Conference on*. 17-19 Aug. 2012. Danang, Vietnam: IEEE, 217–222.

Li, K., Groz, R. and Shahbaz, M. (2006). Integration testing of components guided by incremental state machine learning. In *Testing: Academic and Industrial Conference-Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings*. 29-31 Aug. 2006. Windsor, Canada: IEEE, 59–70.

Lity, S., Baller, H. and Schaefer, I. (2015). Towards incremental model slicing for delta-oriented software product lines. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. 2-6 March 2015. Montreal, QC: IEEE, 530–534.

Liu, C. and Richardson, D. (1998). Software components with retrospectors. In *Proceedings of International Workshop on the Role of Software Architecture in Testing and Analysis*. July 1998. Marsala, Sicily, Italy, 63–68.

Lochau, M., Lity, S., Lachmann, R., Schaefer, I. and Goltz, U. (2014). Delta-oriented model-based integration testing of large-scale systems. *Journal of Systems and Software*. 91, 63–84.

Ma, Y.-S., Oh, S.-U., Bae, D.-H. and Kwon, Y.-R. (2001). Framework for third party testing of component software. In *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*. 4-7 Dec. 2001. Macau, China: IEEE, 431–434.

Machado, P. D. L., Figueiredo, J. C. A., Lima, E. F. A., Barbosa, A. E. V. and Lima, H. S. (2007). Component-based integration testing from UML interaction diagrams. In *2007 IEEE International Conference on Systems, Man and Cybernetics*. 7-10 Oct. 2007. Montreal, Que.: IEEE, 2944–2951.

Mahmood, S. (2011). Towards Component-Based System Integration Testing Framework. In *Proceedings of the World Congress on Engineering (WCE 2011)*. 6-8 July 2011. London, U.K, 1231–1235.

Mahmood, S., Lai, R. and Kim, Y. (2007). Survey of component-based software development. *IET Software*. 1(2), 57–66.

Mahmood, S., Niazi, M. and Hussain, A. (2015). Identifying the challenges for managing component-based development in global software development: Preliminary results. In *Science and Information Conference (SAI)*. 28-30 July 2015. London, UK: IEEE, 933–938.

Mariano, M. M., Souza, E. F., Endo, A. T. and Vijaykumar, N. L. (2016). A comparative study of algorithms for generating switch cover test sets. In *SBQS 2016: 15th Brazilian Symposium on Software Quality*. October 24-26, 2016. Alagoas, Brazil, 6–20.

Martins, E., Toyota, C. and Yanagawa, R. (2001). Constructing self-testable software components. In *Dependable Systems and Networks, 2001. DSN 2001. International Conference on*. 1-4 July 2001. Goteborg, Sweden: IEEE, 151–160.

Merten, M. (2013). *Active automata learning for real life applications*. Ph.D. Thesis. Universität Dortmund.

Meyerer, F. and Hummel, O. (2014). Towards Plug-and-play for Component-based Software Systems. In *Proceedings of the 19th International Doctoral Symposium on Components and Architecture*. 27 June 2014. Marcq-en-Bareul, France: ACM, 25–30.

Mohi-Aldeen, S. M., Mohamad, R. and Deris, S. (2017). Automated path testing using the negative selection algorithm. *International Journal of Computational Vision and Robotics*. 7(1-2), 160–171.

Moiz, S. A. (2017). Uncertainty in Software Testing. In Mohanty, H., Mohanty, J. R. and Balakrishnan, A. (Eds.) *Trends in Software Testing*. (pp. 67–87). Singapore: Springer Singapore.

Momotko, M. and Zalewska, L. (2004). Component+ built-in testing a technology for testing software components. *Foundations of Computing and Decision Sciences*. 29(1-2), 133–148.

Morris, J., Lee, G., Parker, K., Bundell, G. A. and Lam, C. P. (2001). Software component certification. *Computer*. 34(9), 30–36.

Muschevici, R., Proença, J. and Clarke, D. (2015). Feature Nets: behavioural modelling of software product lines. *Software & Systems Modeling*, 1–26.

Mussa, M. and Khendek, F. (2012). Towards a Model Based Approach for Integration Testing. In Ober, I. and Ober, I. (Eds.) *SDL 2011: Integrating System and Software Modeling: 15th International SDL Forum Toulouse, France, July 5-7, 2011. Revised Papers*. (pp. 106–121). Berlin, Heidelberg: Springer Berlin Heidelberg.

Mussa, M. and Khendek, F. (2013). Merging Test Models. In *2013 18th International*

Conference on Engineering of Complex Computer Systems (ICECCS). 17-19 July 2013. Singapore: IEEE, 167–170.

Naseer, F., Rehman, S. and Hussain, K. (2010). Using meta-data technique for component based black box testing. In *Emerging Technologies (ICET), 2010 6th International Conference on*. 18-19 Oct. 2010. Islamabad, Pakistan: IEEE, 276–281.

Neumann, F. (2004). Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. In *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 1. 19-23 June 2004. Portland, OR, USA: IEEE, 904–910.

Neumann, F. (2008). Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. *Computers & Operations Research*. 35(9), 2750–2759.

Niese, O. (2003). *An integrated approach to testing complex systems*. Ph.D. Thesis. Universität Dortmund.

Ning, G., Nakajima, S. and Pantel, M. (2013). Hidden Markov model based automated fault localization for integration testing. In *Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on*. 23-25 May 2013. Beijing, China, 184–187.

Nycander, P. (2015). *Learning-Based Testing of Microservices: An Exploratory Case Study Using LBTest*. Ph.D. Thesis. KTH Royal Institute of Technology.

Orso, A., Do, H., Rothermel, G., Harrold, M. J. and Rosenblum, D. S. (2007). Using component metadata to regression test component-based software. *Software Testing, Verification and Reliability*. 17(2), 61–94.

Orso, A., Jean, M. and Rosenblum, D. (2001). Component metadata for software engineering tasks. In Emmerich, W. and Tai, S. (Eds.) *Engineering Distributed Objects: Second International Workshop,EDO 2000 Davis, CA, USA, November 2–3, 2000 Revised Papers*. (pp. 129–144). Berlin, Heidelberg: Springer Berlin Heidelberg.

Orso, A. and Rothermel, G. (2014). Software Testing: A Research Travelogue (2000-2014). In *Proceedings of the on Future of Software Engineering (FOSE 2014)*. 31 May - 7 June 2014. Hyderabad, India: ACM, 117–132.

Pacharoen, W., Aoki, T., Bhattarakosol, P. and Surarerks, A. (2013). Active Learning of Nondeterministic Finite State Machines. *Mathematical Problems in Engineering*. 2013, 1–11.

Patel, R., Chaudhari, N. and Pawar, M. (2012). Survey of Integrating Testing for Component-based System. *International Journal of Computer Applications*. 57(18),

21–25.

Pramsohler, T., Schenk, S., Barthels, A. and Baumgarten, U. (2015). A layered interface-adaptation architecture for distributed component-based systems. *Future Generation Computer Systems*. 47, 113–126.

Qazi, A. M., Rauf, A. and Minhas, N. M. (2016). A Systematic Review of Use Cases based Software Testing Techniques. *International Journal of Software Engineering and Its Applications*. 10(11), 337–360.

Radhakrishna, A., Lewchenko, N., Meier, S., Mover, S., Sripada, K. C., Zufferey, D., Chang, B.-Y. E. and Černỳ, P. (2017). Learning Asynchronous Typestates for Android Classes. In *Computer Aided Verification, 29th International Conference (CAV 2017)*. January 24, 2017. Heidelberg, Germany: Springer-Verlag, 1–26.

Raffelt, H., Steffen, B. and Berg, T. (2005). Learnlib: A library for automata learning and experimentation. In Baresi, L. and Heckel, R. (Eds.) *Fundamental Approaches to Software Engineering: 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006. Proceedings*. (pp. 62–71). Berlin, Heidelberg: Springer Berlin Heidelberg.

Raffelt, H., Steffen, B., Berg, T. and Margaria, T. (2009). LearnLib: a framework for extrapolating behavioral models. *International journal on software tools for technology transfer*. 11(5), 393–407.

Rehman, M. J. U., Jabeen, F., Bertolino, A. and Polini, A. (2007). Testing software components for integration: a survey of issues and techniques. *Software Testing Verification & Reliability*. 17(2), 95–133.

Reid, S. C. (2000). BS 7925-2: the software component testing standard. In *Quality Software, 2000. Proceedings. First Asia-Pacific Conference on*. 30-31 Oct 2000. Hong Kong, China: IEEE, 139–148.

Reza, H. and Cheng, L. (2012). Context-Based Testing of COTs Using Petri Nets. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*. 16-18 April 2012. Las Vegas, NV: IEEE, 572–577.

Rivest, R. L. and Schapire, R. E. (1993). Inference of finite automata using homing sequences. *Information and Computation*. 103(2), 299–347.

Rösch, S., Ulewicz, S., Provost, J. and Vogel-Heuser, B. (2015). Review of Model-Based Testing Approaches in Production Automation and Adjacent Domains - Current Challenges and Research Gaps. *Journal of Software Engineering and Applications*. 8(9), 499–519.

Sandin, E. V., Yassin, N. M. and Mohamad, R. (2017). Comparative Evaluation of Automated Unit Testing Tool for PHP Framework. *International Journal of Software Engineering and Technology*. 2(2), 7–11.

Schach, S. R. (2001). *Object-Oriented and Classical Software Engineering*. (5th ed.). United States: McGraw-Hill Pub. Co.

Schieferdecker, I., Dai, Z. R., Grabowski, J. and Rennoch, A. (2003). The UML 2.0 testing profile and its relation to TTCN-3. In Hogrefe, D. and Wiles, A. (Eds.) *Testing of Communicating Systems: 15th IFIP International Conference, TestCom 2003, Sophia Antipolis, France, May 26–28, 2003 Proceedings*. (pp. 79–94). Berlin, Heidelberg: Springer Berlin Heidelberg.

Schneidewind, N. (2009). Integrating testing with reliability. *Software Testing, Verification & Reliability*. 19(3), 175–198.

Shahbaz, M. (2008). *Reverse engineering enhanced state models of black box software components to support integration testing*. Ph.D. Thesis. Grenoble Institute of Technology.

Shahbaz, M. and Groz, R. (2009). Inferring mealy machines. In Cavalcanti, A. and Dams, D. R. (Eds.) *FM 2009: Formal Methods: Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*. (pp. 207–222). Berlin, Heidelberg: Springer Berlin Heidelberg.

Shahbaz, M. and Groz, R. (2014). Analysis and testing of black-box component-based systems by inferring partial models. *Software Testing, Verification and Reliability*. 24(4), 253–288.

Shahbaz, M., Li, K. and Groz, R. (2007). Learning Parameterized State Machine Model for Integration Testing. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*. 24-27 July 2007. Beijing, China: IEEE, 755–760.

Shang, X. and Zhang, Y. (2006). Research of UML-based generating test case for component integration testing. *Computer Engineering*. 32, 96–98.

Shashank, S., Chakka, P. and Kumar, D. (2010). A systematic literature survey of integration testing in component-based software engineering. In *Computer and Communication Technology (ICCCT), 2010 International Conference on*. 17-19 Sept. 2010. Allahabad, Uttar Pradesh: IEEE, 562–568.

Shu-Fen, L., Yi-Kun, Z., Jing, Z. and Wen-Yuan, X. (2010). Based on Markov Process Method for Integration Testing. In *The Third International Symposium on Electronic Commerce and Security Workshops (ISECS 2010)*. 29-31 July 2010. Guangzhou, China, 211–214.

Shukla, R. and Marwala, T. (2012). Component Based Software Development Using Component Oriented Programming. In Kumar M., A., R., S. and Kumar, T. V. S. (Eds.) *Proceedings of International Conference on Advances in Computing*. (pp. 1125–1133). New Delhi, India: Springer India.

Siddiqui, Z. A. and Tyagi, K. (2016). Application of fuzzy-MOORA method: Ranking of components for reliability estimation of component-based software systems. *Decision Science Letters*. 5(1), 169–188.

Silva, F. R. C., Almeida, E. S. and Meira, S. R. L. (2009). An Approach for Component Testing and Its Empirical Validation. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC '09)*. 9-12 March 2009. Honolulu, Hawaii: ACM, 574–581.

Sirohi, N. and Parashar, A. (2013). Component Based System and Testing Techniques. *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*. 2(6), 2378–2383.

Sneed, H. M. (2010). Testing Object-oriented Software Systems. In *Proceedings of the 1st Workshop on Testing Object-Oriented Systems (ETOOS '10)*. 21-25 June 2010. Maribor, Slovenia: ACM, 161–165.

Stefanescu, A., Wendland, M.-F. and Wieczorek, S. (2010). Using the UML testing profile for enterprise service choreographies. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*. 1-3 Sept. 2010. Lille, France: IEEE, 12–19.

Suman, R. R., Mall, R., Sukumaran, S. and Satpathy, M. (2010). Extracting State Models for Black-Box Software Components. *Journal of Object Technology*. 9(3), 79–103.

Tao, C., Gao, J. and Li, B. (2015). A Model-Based Framework to Support Complexity Analysis Service for Regression Testing of Component-Based Software. In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*. March 30 2015 - April 3 2015. San Francisco Bay, CA: IEEE, 326–331.

Tarawneh, F., Baharom, F., Yahaya, J. H. and Ahmad, F. (2011). Evaluation and selection COTS software process: the state of the art. *International Journal of New Computer Architectures and their Applications (IJNCAA)*. 1(2), 344–357.

Tiwari, A. and Chakraborty, P. S. (2015). Software Component Quality Characteristics Model for Component Based Software Engineering. In *Computational Intelligence & Communication Technology (CICT), 2015 IEEE International Conference on*. 13-14 Feb. 2015. Ghaziabad, Uttar Pradesh: IEEE, 47–51.

Tiwari, U. K. and Kumar, S. (2017). Components integration-effect graph: a black

box testing and test case generation technique for component-based software. *International Journal of System Assurance Engineering and Management*. 8(2), 393 – 407.

Tran, H.-V., Le, C.-L., Nguyen, Q.-T. and Ngoc Hung, P. (2015). An Efficient Method for Automated Generating Models of Component-Based Software. In Nguyen, V.-H., Le, A.-C. and Huynh, V.-N. (Eds.) *Knowledge and Systems Engineering*. (pp. 499–511). Cham: Springer International Publishing.

Utting, M., Pretschner, A. and Legeard, B. (2012). A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*. 22(5), 297–312.

Vaandrager, F. (2017). Model learning. *Communications of the ACM*. 60(2), 86–95.

Vale, T., Crnkovic, I., de Almeida, E. S., Neto, P. A. d. M. S., Cavalcanti, Y. C. and de Lemos Meira, S. R. (2016). Twenty-eight years of component-based software engineering. *Journal of Systems and Software*. 111, 128–148.

Verma, D. (2012). Component Testing Using Finite Automata. *Indian Journal of Computer Science and Engineering (IJCSE)*. 3(5), 658–666.

Voas, J. M. (1998). Certifying off-the-shelf software components. *Computer*. 31(6), 53–59.

Wang, Y., King, G. and Wickburg, H. (1999). A method for built-in tests in component-based software maintenance. In *Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on*. 3-5 March 1999. Amsterdam, The Netherlands: IEEE, 186–189.

Wu, Y., Chen, M. H. and Offutt, J. (2003). UML-based integration testing for component-based software. In Erdogmus, H. and Weng, T. (Eds.) *COTS-Based Software Systems: Second International Conference, ICCBSS 2003 Ottawa, Canada, February 10–12, 2003 Proceedings*. (pp. 251–260). Berlin, Heidelberg: Springer Berlin Heidelberg.

Wu, Y., Pan, D. and Chen, M.-H. (2001). Techniques for testing component-based software. In *Engineering of Complex Computer Systems, 2001. Proceedings. Seventh IEEE International Conference on*. 11-13 June 2001. Skovde, Sweden: IEEE, 222–232.

Xie, T. and Notkin, D. (2003). Exploiting synergy between testing and inferred partial specifications. In *In WODA, 2003 ICSE Workshop on Dynamic Analysis*. 2 April 2003. Portland, Oregon, 17–21.

Zaki, M. Z., Jawawi, D. N. and Isa, M. A. (2015). Integrated MARTE-based Model for Designing Component-Based Embedded Real-Time Software. *International*

Journal of Software Engineering and Its Applications. 9(3), 157–174.

Zheng, W. and Bundell, G. (2008). Test by Contract for UML-Based Software Component Testing. In *Computer Science and its Applications, 2008. CSA '08. International Symposium on*. 13-15 Oct. 2008. Hobart, ACT: IEEE, 377–382.

Zheng, W. Q. and Bundell, G. (2007a). Model-based software component testing: A UML-based approach. In *6th IEEE/ACIS International Conference on Computer and Information Science, Proceedings*. 11-13 July 2007. Melbourne, Qld.: IEEE, 891–898.

Zheng, W. Q. and Bundell, G. (2007b). A UML-based methodology for software component testing. In *IMECS 2007: International Multiconference of Engineers and Computer Scientists*. 21-23 March 2007. Hong Kong, China, 1177–1182.